A MAXIMAL FIGURE-OF-MERIT LEARNING APPROACH TO TEXT CATEGORIZATION

Sheng Gao[a,b], Wen Wu[c], Chin-Hui Lee[c,d] and Tat-Seng Chua[c]

[b]Institute for Infocomm Research
21 Heng Mui Keng Terrace
Singapore 119613
gaosheng@i2r.a-star.edu.sg

[c]School of Computing
National Univ. of Singapore
3 Science Drive 2, Singapore, 117543
{wuwen,chl,chuats}@comp.nus.edu.sg

[d]School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA USA
chl@ece.gatech.edu

ABSTRACT

A novel maximal figure-of-merit (MFoM) learning approach to text categorization is proposed. Different from the conventional techniques, the proposed MFoM method attempts to integrate any performance metric of interest (e.g. accuracy, recall, precision, or F_1 measure) into the design of any classifier. The corresponding classifier parameters are learned by optimizing an overall objective function of interest. To solve this highly nonlinear optimization problem, we use a generalized probabilistic descent algorithm. The MFoM learning framework is evaluated on the Reuters-21578 task with LSI-based feature extraction and a binary tree classifier. Experimental results indicate that the MFoM classifier gives improved F_1 and enhanced robustness over the conventional one. It also outperforms the popular SVM method in micro-averaging F_1. Other extensions to designing discriminative multiple-category MFoM classifiers for application scenarios with new performance metrics could be envisioned too.

1. INTRODUCTION

Text categorization (TC) is a process of classifying a text document into some pre-defined categories. It is an important research problem in information retrieval (IR), information extraction and filtering and natural language processing. In the past two decades TC has received much attention [32]. A number of machine learning approaches to TC have been proposed. They are Bayesian method [20,23,25], *k*-nearest neighbors (*k*NN) [24,36,37], Rocchio algorithm [5,12], artificial neural networks (ANN) [10,22,27,29,31], support vector machines (SVM) [6,13,14], boosting [30], decision tree (DT) [3,4,10,20,28,33], and hidden Markov model (HMM) [8,26]. Some are based on widely available software packages, such as C4.5 [1], CART[2], and SVM[3].

Most of these methods aim at learning parameters of a joint distribution, *P(X,C)*, between the observed feature, *X*, of a document and its corresponding category, *C*. It is well known that if *P(X,C)* is specified exactly, an optimal classifier can be designed to minimize the Bayes risk (e.g. [18]). Unfortunately, for most real-world problems, the exact form of the joint distribution is usually unavailable. Even we are lucky to be given the distribution form, the parameters of the joint distribution would still have to be estimated from a labeled training set. To reduce the complexity of the design, *P(X,C)* is usually decomposed into two components, *P(X|C)* and *P(C)*, known as the conditional class and prior class distributions, respectively. In most pattern recognition scenarios, the latter is often assumed to be equally probable and ignored [20,23,25]. The former is assumed to be a distribution with a known form, e.g. mixture Gaussian densities in HMM [19], multinomial density in naïve Bayes [20,23,25], etc. *Maximum likelihood* (ML) techniques are then used to estimate the parameters. When the actual training and testing data do not support this assumption, the system performance is degraded. To overcome this mismatch for any given classifier with any density, *minimum classification error* (MCE) techniques [15,16,17,18] were proposed to directly minimize the empirical error of the training set. A family of *generalized probabilistic descent* (GPD) [16] algorithms was used to solve for the parameters in an iterative manner. The MCE framework has been successfully adopted in speech and speaker recognition and related problems [15,16]. Recently it was extended to vector-based call routing [17].

For TC, the conventional techniques do not directly train the parameters of the corresponding classifier based on the criterion of maximizing a real performance metric, such as recall, precision, or F_1 measure. Furthermore, it becomes flexible for most researchers if appropriate metrics could be chosen for different tasks and a system could then be designed accordingly.

In this paper we propose a novel *Maximal Figure-of-Merit* (MFoM) approach that has a consistent learning objective with the evaluation metrics of a given application. The difficulty in such a formulation lies in computing discrete quantities, such as recall, precision and F_1, because they are not differentiable functions of the parameters and therefore could not be easily optimized. In the proposed framework, we design a training objective which directly relates the classifier parameters to the performance metric of interest through a smooth approximation of the errors and related metrics that could be optimized directly.

[a] This work was done when the first author was with NUS.
[1] http://www.cse.unsw.edu.au/~quinlan/
[2] http://www.salford-systems.com/
[3] http://www.ece.umn.edu/groups/ece8591/software/svm.html

The remainder of the paper is organized as follows. Section 2 discusses TC work related to this study. Section 3 describes document feature extraction based on latent semantic indexing (LSI). Section 4 introduces theory of the MFoM learning approach. Section 5 lays out the iterative GPD algorithm. Experimental results based on the Reuters-21578 task are then presented in Section 6. Finally we summarize our findings in Section 7.

2. RELATED WORK IN TC

A good tutorial on the state-of-the-art of TC techniques can be found in [32]. For work relevant to this study we focus more discussion on decision trees [3,4,10,28,33] and SVM [6,13,14] because binary tree classifiers are our baseline classifiers and SVM is the target classifier with which we compare our proposed MFoM-based approaches. Details on other machine learning methods have also been documented [8,20,21,22,23,24,25,26,27,30,37].

2.1 Decision Tree (DT)

Given a set of labeled training samples each of which is represented by a feature vector (the vector component could either be a numeric or symbolic value), a regression tree could be constructed by recursive partitioning of samples that belong to a tree node. At each node, a decision rule that minimizes an impurity function is chosen to partition the training samples. To avoid over-fitting, tree pruning is needed. Cross-validation can also be applied if necessary. The decision rule at each node can either be a non-parametric or parametric model. A non-parametric rule could be a statement, like "*if feature X < 5.0?*" or "*Y==male?*". A parametric example could be a linear discriminatnt function [4,33], a neural network [10], the Rocchio algorithm or Naïve Bayesian [35]. Some DT tools, such as CART, ID3 and C4.5, have been widely adopted [3,28] for TC applications.

2.2 Binary Tree Classifier

TC is often solved by designing a set of *N binary classifiers*, each only determines if a document is relevant o to a specific category. Each binary classifier is trained from a category-based training set, which is obtained by re-labeling each document in the training corpus T with either the positive class, $C+$, if it is relevant to the category, or the negative class, C-, if irrelevant. Figure 1 shows an example. Each node, except the root, is given a relevance label (positive or negative). All training samples that enter into it are classified according to the node label. Recursive splitting is then performed to build the tree node-by-node.

For each node a decision rule needs to be defined. In this study we use a linear discriminant function (LDF) which can be learned from these samples corresponding to the node. For a feature vector *X*, we define the LDF as

$$f(W,X)=\sum_{k=1}^{R} w_k x_k - w_0 , \qquad (2.1)$$

where R is the feature dimension, W is the weight vector, and $x_i\,(i=1,2,\Lambda\ R)$ is the *i*-th component of *X*.

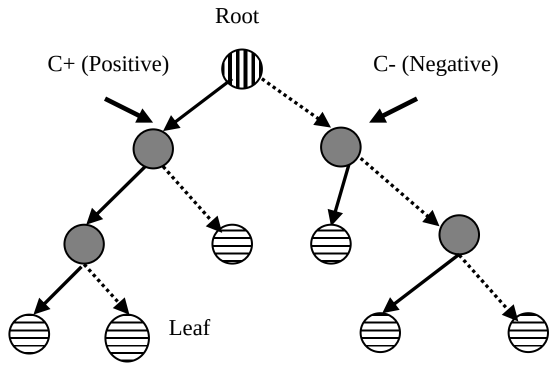

Figure 1 A Binary Tree Classifier

In order to split a set of samples into two child nodes, called *Yes* node labeled by the positive class (left branch of a node in Figure 1) and *No* node labeled by the negative (right branch of a node in Figure 1), a decision rule is applied. For any document with feature, X, if $f(W,X)>0$, it is assigned into the *Yes* node (C+). Otherwise, it is assigned into the *No* node (C-). The optimal parameters W are learned using a number of algorithms to be discussed later. The above procedure is run recursively until there are no more nodes left to be split. At this point a binary tree classifier is finally established.

In testing for a given document with feature *X*, the learned tree classifier assigns it as either positive or negative. The process starts from the root. If $f(W,X)>0$, it enters the *Yes* node (left branch). Otherwise, it enters the *No* node. It continues until a leaf is reached, the label at the leaf is then assigned to the document.

2.3 Support Vector Machine (SVM)

The support vector machine learning framework was first proposed to solve the two-class classification problem by Vapnik in 1979 (in Russian). The decision function is defined as $z_k f(W,X_k)\geq 1,\ \mathrm{k}=1,\Lambda\ ,\mathrm{K}$, where $z_k=1$ stands for positive and $z_k=-1$ for negative cases, W is the parameters to be estimated, X_k is the k-th sample vector, and *K* is the total number of the training samples.

SVM tries to find a decision surface which can maximally separate two classes based on the structural risk minimization principle [34]. Similar to the binary tree classifier discussed above with LDF, a linear decision function, $f(W,X_k)=W\cdot X_k-b$, is used in the case of linear SVM. The parameter vector, *W*, could be estimated by minimizing $\|W\|^2$ subject to the constraint, $z_k(W\cdot X_k-b)\geq 1, \forall k$.

Joachims [13,14] applied SVM to TC. His experimental results showed that SVM outperformed most existing TC classifiers, such as Naïve Bayesian, Rocchio algorithm, C4.5, and *k*-NN.

3. LSI-BASED FEATURE EXTRACTION

In IR, a document is often represented by a feature vector with the dimension equal to the size of the lexicon. Each component of the feature vector corresponds to the contribution of a term occurred in the document. In a typical application the lexicon usually has more than ten thousands entries. Many techniques, such as feature selection [32], have been proposed to reduce the dimension. Latent semantic indexing (LSI) [1] is a way to achieve both feature extraction and reduction. Probabilistic LSI (PLSI) [11] has also been used. In this study, singular value decomposition (SVD) based LSI is used to get a lower dimension than the original one by decomposing the term-document matrix *H* into a multiplication of three matrices:

$$H = USV^T \quad , \tag{3.1}$$

$U : M\times R$ left singular matrix with rows $u_i, 1 \le i \le M$,

$S : R\times R$ diagonal matrix of singular values $s_1 \ge s_2 \ge .. \ge s_R > 0$;

$V : N\times R$ right singular matrix with rows $v_j, 1 \le j \le N$.

Both left and right singular matrices are column-orthonormal. If we retain only the top *Q* singular values in matrix *S* and zero out the other (*R*-*Q*) components, the LSI feature dimension could be effectively reduced to *Q* which could be much smaller than *R*. By doing so, the three matrices are much smaller in size than those in Eq. (3.1) and it greatly reduces the computation requirements. We will explore some possibilities in Section 6.

4. MFoM LEARNING APPROACH

As discussed above it is impossible to design an optimal Bayes classifier because it is unlikely to know the exact joint distribution, *P(X,C)*, between the feature, *X*, and the category, *C*. In this study we propose a maximal figure-of-merit (MFoM) learning approach to optimize any given performance metric for any given classifier directly. Since most evaluation metrics of interest in classification are discrete functions of error counts and not differentiable, some smooth approximation is needed to embed given classifiers into overall objective functions of these metrics.

4.1 Performance Metrics

A set of metrics is needed to evaluate the performance of classification systems. To achieve the best performance for different applications with different requirements, it is important to take into account the effect of these metrics on the design of the classifier.

Denote the precision, recall, and F$_1$ measure for a class C_j by P_j, R_j, and F_j , respectively. They are defined as

$$P_j = \frac{TP_j}{TP_j + FP_j} \tag{4.1}$$

$$R_j = \frac{TP_j}{TP_j + FN_j} \tag{4.2}$$

$$F_j = \frac{2P_jR_j}{R_j + P_j} = \frac{2TP_j}{FP_j + FN_j + 2TP_j} \tag{4.3}$$

TP_j (true positive): no. of documents correctly assigned;

FP_j (false positive): no. of documents falsely accepted;

FN_j (false negative): no. of documents falsely rejected.

It is clear that these metrics are discrete quantities for counting errors and could not be optimized directly because they are not differentiable functions of the parameters.

4.2 Approximating Overall Performance Metrics

Define a class loss function, $l_j(X;W)$, for class C_j. To approximate the error counts its value should be close to 0 for correct, and 1 for incorrect classification, respectively. Clearly this loss is a function of the feature vector, *X*, and the classifier parameters, *W*. Then the true positive, false positive and false negative functions for C_j, summing over all *K* samples in *T*, could be approximated as follows:

$$TP_j \approx \sum_{X\in T} \left(1 - l_j(X;W)\right)\cdot 1\left(X \in C_j\right) \tag{4.4}$$

$$FP_j \approx \sum_{X\in T} l_j(X;W)\cdot 1\left(X \notin C_j\right) \tag{4.5}$$

$$FN_j \approx \sum_{X\in T} l_j(X;W)\cdot 1\left(X \in C_j\right) \tag{4.6}$$

where *1(A)* is the indicator function of any logical expression, *A*. As for the choice of an appropriate loss function, any smooth 0-1 function of a one dimension variable approximating a step function at the origin will do the job. A sigmoid function shown below is often adopted [15,16,17,18]

$$l_j(X;W) = \frac{1}{1 + e^{-(\alpha d_j(X;W) + \beta)}} \quad , \tag{4.7}$$

where α is a positive constant that controls the size of the learning window and the learning rate, and β is a constant measuring the offset of $d_j(X;W)$ from 0. The most crucial step in the embedding process is to define a class misclassification function, $d_j(X;W)$, that is negative for a correct decision and positive in value otherwise. In the binary tree classifiers, with the LDF node decision rules shown in Eq. (2.1), we have the following natural choice:

$$\begin{cases} d_j(X;W) = -f(X,W) = -\sum_{k=1}^{R} w_k x_k + w_0 & \text{for C}+ \\ d_j(X;W) = f(X,W) = -w_0 + \sum_{k=1}^{R} w_k x_k & \text{for C}- \end{cases} \quad . \tag{4.8}$$

For a single level, binary tree classifier the measure in Eq. (4.8) serves as a simple way to embed the classifier into the performance metrics of interest. However in the multi-level binary tree classifiers we are designing, there is no simple way of defining an overall misclassification measure in a global manner. Instead we repetitively use the function in Eq. (4.8) in every node of the tree. This results in the need for optimizing the node level performance metrics in a recursive manner. Although local optimization at each individual node does not necessarily lead to global optimization of the entire tree, such a node-by-node optimization of the performance metrics have produced satisfactory results just like in the case of designing of multi-level binary tree classifiers.

Now we are ready to formulate an overall objective function to serve as a figure-of-merit for an application. For example, using the class F_1 measure approximation defined in Eq. (4.3), we could easily define an overall F_1 measure approximation as follows (N : the number of classes):

$$F(T;W)=\frac{1}{N}\sum_{j=1}^{N}F_j=\frac{1}{N}\sum_{j=1}^{N}\frac{2TP_j}{FP_j+FN_j+2TP_j}. \quad (4.9)$$

By maximizing the F_1 measure of each individual class, we have an optimal set of N binary classifiers that maximizes the overall F_1 measure as defined in Eq. (4.9). Another way is to define the following approximated error rate at each node for each class using the MCE objective (L : the number of samples in T),

$$L(T;\Lambda)=\frac{1}{L}\sum_{X\in T}\sum_{j=1}^{N}l_j(X;\Lambda)1(X\in C_j) \quad . \quad (4.10)$$

4.3 Multiple-Category Classifiers and MFoM

We now extend the discussion on binary classifier to more complex cases. We will show below that the MFoM learning framework is ideal for designing discriminative multi-category classifiers. For a decision rule that classifies a given document X into one of N categories, a popular choice is to maximize over all class scores, i.e.

$$C(X)=\arg\max_j g_j(X;\Lambda), 1\le j\le N, \quad (4.11)$$

where $C(X)$ is the class label assigned by the decision rule, $g_j(X;\Lambda)$ is a class discriminant function to compute class scores, and Λ the entire set of the parameters of the classifiers. Such rules have been used extensively in other applications, such as automatic speech recognition in which one out of many possible sentences are chosen as the most likely recognized string. This of course could not be easily accomplished by binary classifiers. In multi-category classification, a correct classification decision is made for a document X coming from the class C_j, if $C(X)=C_j$, i.e. $\forall i$

$$g_j(X;\Lambda)>g_{i\ne j}(X;\Lambda) \quad X\in C_j. \quad (4.12)$$

One way to embed the discrete decision rule in Eq. (4.11) into an optimization objective function is to define a one-dimension class misclassification function, $d_j(X;\Lambda)$, such that Eq. (4.12) is equivalent to $d_j(X;\Lambda)<0$. This could be accomplished as follows [15,16,17,18] by:

$$d_j(X;\Lambda)=-g_j(X;\Lambda)+\left[\frac{1}{N-1}\left\{\sum_{i,i\ne j}g_i^{\eta}(X;\Lambda)\right\}\right]^{1/\eta}, \quad (4.13)$$

where η is a positive constant, N is the total number of categories, and the second term is a geometric average of all the scores from other competing classes. Intuitively it is noted that when η approaches ∞, the second term of the RHS of Eq. (4.13) converges to the highest score among the competing ones. The absolute value of the LHS in Eq. (4.13) quantifies the separation between the correct and the competing classes. We could maximize this separation by minimizing the expected value of $d_j(X;\Lambda)$ or any non-decreasing function of it. This enhances the robustness and improves the classification accuracy. Again the same loss function, as defined in Eq. (4.7), could be used to approximate the error counts and therefore any function of them. For MCE, as an illustration, the total error could be approximated similarly as shown in Eq. (4.10). This is also the overall objective function to be optimized. GPD algorithms could then be used to find the solution. There are no other known formulations capable of directly solving this type of discriminative multi-category classification.

5. LEARNING ALGORITHMS

5.1 Learning Baseline Binary Tree Classifiers

Now let us give a brief discussion on our baseline binary tree classifier. Many algorithms were proposed to find the weights of LDF [3,4,28,33]. In this paper, we use the traditional gradient descent algorithm based on the perceptron criterion function *L(T;W)* defined below to train our baseline classifier [9] at each tree node,

$$L(T;W)=\sum_{X\in T}-f(W,X') \quad , \quad (5.1)$$

where X' is an augmented feature vector with an element with value 1 padded to the original *X*. By minimizing the perceptron criterion the parameters can be updated iteratively using the batch perceptron algorithm (BPA) [9],

$$W_{t+1}=W_t+\varepsilon_t\cdot\sum_{X\in T_t}X' \quad . \quad (5.2)$$

where T_t is the set of the incorrectly classified samples and and ε_t the learning constant at the *t*-th iteration.

5.2 Learning MFoM Binary Tree Classifiers

Now let us discuss the detailed algorithm that learns the parameters W of the LDF using the MFoM approach. At each node we adopt the class misclassification function defined for binary classification in Eq. (4.8).

Generally the above defined overall objective function is a highly nonlinear function of the parameters *W* of the classifier. It is difficult or impossible to find its closed solution. To solve the optimization problem, GPD algorithm is employed to seek the optimal *W** that can minimize *L(T;W)=-F(T;W)* defined in Eq. (4.9). From the above definitions, the overall objective function is a continuous and differentiable function. Denote $\nabla L(T;W)$ as its gradient. Then *W** can be found using an iterative method, which has been proved to converge [15] to a local minimum,

$$W_{t+1} = W_t - \kappa_t \nabla L(T;W)|_{W=W_t} \quad , \tag{5.3}$$

with W_t being the parameters at the t-th iteration, and κ_t being an update step size or learning rate. It is interesting to note the similarity between Eqs. (5.3) and (5.2). The components of the gradient vector of the overall objection $\partial L(X;W)/\partial w_i$ in Eq. (5.3) can be computed using the chain rules. Here we skip detail of the derivation.

5.3 Parameter Initialization

The classifier parameters should be initialized properly to reduce training time and speed up the convergence. Here the weights, $(w_1, w_2, \Lambda\ , w_R)$, are set equal to the mean of the training samples for the class with the minimal number of the training documents, while the offset term w_0 is initialized at the minimum value of the function $\sum w_k x_k$.

The above initialization strategy works for both BPA and GPD learning. For GPD initialization it is also fine to use the parameter set of the baseline classifier obtained with BPA. In practice, both methods work reasonably well.

6. RESULTS AND ANALYSIS

6.1 Experimental Setup

In the following we reported our experiments with the ModApte version of the Reuters-21578 TC task[4]. Many evaluations have been documented [8,14,20,22,35,36]. The original format of the text documents is in SGML. We perform some preprocessing to filter out the unused parts of a document. We preserved only the title, dateline, and the body text, and removed unlabeled documents. We then retained only the categories that have at least one document in the training and the testing sets. This process results in a collection of 90 categories for training and testing. 7,770 training and 3,019 testing documents are left. A set of 319 stop words and terms that occur less than 4 times are removed. This gave a lexicon with 10,118 entries. Document distributions over the 90 categories in both the training and the testing set are very unbalanced. For example, the most frequent category 'earn' has 2,877 training documents. On the other hand some categories, such as 'sun-meal', 'castor-oil', 'lin-oil', have only one document. This results in inconsistency in comparison.

To evaluate the classification performance for each category, precision, recall, and F_1 measure in Eqs. (4.1)-(4.3) are used. To evaluate the average performance for the 90 categories, the macro-averaging F_1 and micro-averaging F_1 are used [32,36] and defined as follows:

$$F_1^{\,M} = \frac{2\sum_{i=1}^{N} R_i \sum_{i=1}^{N} P_i}{N\left(\sum_{i=1}^{N} P_i + \sum_{i=1}^{N} R_i\right)}, \tag{6.1}$$

$$F_1^{\,\mu} = \frac{2\sum_{i=1}^{N} TP_i}{\sum_{i=1}^{N} FP_i + \sum_{i=1}^{N} FN_i + 2\sum_{i=1}^{N} TP_i}, \tag{6.2}$$

i.e. micro-averaging method calculates the global measures by giving category's local performance measures different weights based on their numbers of positive documents. Macro-averaging method treats every category equally, and calculates global measures as the mean of all categories' local measures. It is clear that the overall objective function defined in Eq. (4.9) resembles the micro-averaging F_1 defined in Eq. (6.2). It will be shown later that the F_1-based MFoM approach formulated accordingly also works better when using micro-averaging F_1 for comparison.

6.2 Experimental Results

A 10,118x7,770 term-document matrix was first built from the training set using the LSI feature extraction method as discussed in Section 3. After SVD[5] [2], the rank is found to be 1,613, the maximal dimension of the feature vector in the latent semantic space. Compared to the original dimension there is a substantial reduction even though no further feature reduction is yet imposed.

In all the following experiments of MFoM learning, the parameter κ_0 in Eq. (5.3) is set to 0.05. For the Sigmoid function parameters in Eq. (4.7), α is assumed to be fixed at 20, and β is dynamically adjusted and set to be the average of the class misclassification function values of all the training samples in each iteration. The exact parameter values are not crucial for the experimental results.

6.2.1 MCE vs. Maximum F_1 MFoM Learning

One advantage of MFoM learning is the ability to integrate any performance metric of interest into an overall objection function and to learn the parameters of the classifier by optimizing this function. In Section 4, we have introduced two kinds of the overall objective function, one based on the error rate and another on the F_1 measure. In Table 1 we compare the performances of the two methods for the top 10 and the other 80 categories using two feature sets, a reduced dimension of 400 and the full rank of 1,613. For most of the top 10 categories the F_1-based MFoM gives a better performance when compared with the MCE-based MFoM (except for 2 categories, 'grain' and 'corn'). For the other 80 categories, the comparison shows mixed results. Since we use the F_1-based MFoM learning objective to simulate the micro-averaging F_1 as defined in Eq. (6.1) and to obtain the results listed in Table 1, it seems to indicate that the micro-averaging F_1 is a more consistent measure than the macro-averaging F_1 for comparison purposes.

[4] http://www.research.att.com/~lewis/reuters21578.html

[5] http://www.netlib.org/svdpack/

Table 1 Macro-averaging F_1 and micro-averaging F_1

Feature dimension		400		1,613	
Merit-of-figure		MCE	F_1	MCE	F_1
Top-10	micro-avg	0.9099	0.9273	0.9157	0.9307
	macro-avg	0.8474	0.8728	0.8890	0.8778
Other 80	micro-avg	0.6548	0.6770	0.7007	0.7141
	macro-avg	0.5234	0.4849	0.5659	0.5124

Next we study the effect of the LSI feature dimension on the performance. We vary the dimension from 100 to its full rank of 1,613, and the iteration number from 500 to 3,000. The results are shown in Tables 2 and 3 for the macro-averaging and micro-averaging F_1, respectively.

Table 2 Macro-averaging F_1 (for all 90 categories) as a function of LSI feature dimensions and GPD iterations

Dim	500ite	1000ite	2000ite	3000ite
100	0.4720	0.4642	0.4643	0.4612
200	0.5042	0.5061	0.5040	0.5050
400	0.5318	0.5324	0.5304	0.5280
800	0.5410	0.5395	0.5404	0.5384
1200	0.5556	0.5660	0.5645	0.5650
1613	0.5557	0.5550	0.5560	0.5540

Looking at each column in both tables, we can see that both the macro-averaging F_1 and micro-averaging F_1 values increase in most cases when the dimension of the LSI feature varies from 100 until its full rank of 1,613. When the dimension is beyond 800, only little improvement is observed. The values in bold font in each column of Table 3 indicate the best F_1 for a given LSI feature dimension with a fixed number of iterations. Since GPD only attains a local optimum in a probabilistic manner we could only set the maximum number experimentally. In principle, we want to choose a small value to speed up training but not too small in order to achieve a fast convergence. Other faster algorithm have been studied but not reported here.

Table 3 Micro-averaging (for all 90 categories) as a function of LSI feature dimensions and GPD iterations

Dim	500ite	1000ite	2000ite	3000ite
100	0.8325	0.8264	0.8265	0.8254
200	0.8541	0.8558	0.8541	0.8535
400	0.8702	0.8723	0.8696	0.8697
800	0.8780	0.8793	0.8817	0.8801
1200	0.8802	0.8817	0.8819	0.8825
1613	**0.8809**	**0.8822**	**0.8842**	**0.8826**

6.2.2 Comparing F_1-based MFoM, Baseline and SVM

For TC, SVM classifiers give the best performance in most of the data sets. For the Reuters-21578 task, SVM overwhelms the other classifiers. In this subsection, we compare our binary tree classifier learned by F_1-based MFoM (2000 GPD iterations) with the baseline classifier learned by the traditional gradient descent algorithm based on the perceptron criterion function and linear SVM (C=1.0, which achieves the best performance according to the micro-averaging F_1 on all 90 categories) [13]. Here, the baseline classifier and MFoM tree classifier both use the full rank LSI feature. SVM uses about 9,600 features without feature reduction. Their performance comparison is shown in the Table 4.

Table 4 Performance comparison in F_1 among the baseline binary tree, linear SVM and MFoM classifiers

Category	Baseline	Linear SVM	MFoM
Earn	0.979	0.982	0.979
Acq	0.953	0.956	0.968
Money-fx	0.784	0.785	0.826
Grain	0.889	0.931	0.906
Crude	0.887	0.894	0.897
Trade	0.730	0.792	0.807
Interest	0.743	0.748	0.792
Ship	0.853	0.865	0.878
Wheat	0.829	0.868	0.870
Corn	0.821	0.878	0.891
Micro-avg (all 90)	0.854	0.875	0.884
Macro-avg (all 90)	0.519	NA	0.556

In the top 10 categories, the F_1 measure with MFoM is slightly better than that with SVM, and mcuh better than the baseline. There are only two categories (‘earn’ and ‘grain’) in which linear SVM showed better performance than MFoM. It is noted from Table 3 and 4 that even using only 400 features, F_1-based MFoM learning gives a micro-averaging F_1 value of 0.872, comparable with the value of 0.875, obtained with linear SVM using all 9,600 features.

6.2.3 Properties of MFoM Learning Method

We now give some analysis of the MFoM learning algorithm and study some nice properties. They can partly explain its success in TC. As we have discussed, the optimal LDF weights are learned with the GPD algorithm by maximizing the F_1 measure of the positive class. Figure 2 shows the convergence property of the GPD algorithm for category ‘acq’ using “equal weight” initialization. Only at the beginning of the process (less than about 50 iterations in this figure), there are some fluctuation of F_1 values because the learning rate of GPD was set large initially and reduced linearly after running a few iterations. The F_1 measure increases smoothly from 0.362 until it reaches a stable value of 0.960 after about 150 iterations. It is also interesting to note that this converging value for the training set is similar to the value of 0.968 for testing as shown in Table 4, a close prediction of actual performance.

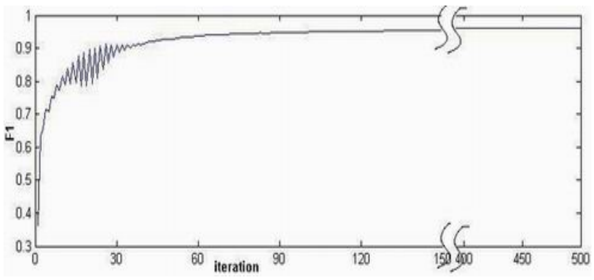

Figure 2 GPD convergence for category 'acq' (feature dimension: 400, X-axis: number of the iteration, Y-axis: F_1 measure for the positive class over training samples)

Figure 3 shows the four distributions (represented by the four histograms) of the class misclassification function values for the positive and negative classes for the category 'acq' in the training set. 2 curves show the distributions at the beginning of GPD, and 2 for the distributions after 500-iteration of F_1-based MFoM learning. The arrows in the figure indicate them respectively. An improved separation between positive and negative classes over the original classifier is clearly observed after MFoM training.

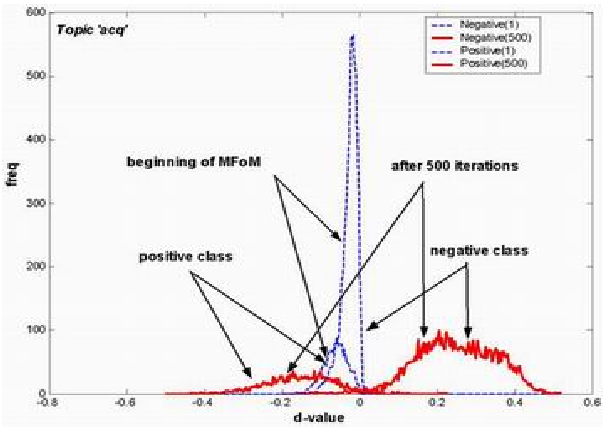

Figure 3 d-value distributions before and after F_1-based MFoM training for category 'acq' (500 iterations, X-axis: d-value, Y-axis: frequency count)

As we pointed out in Section 4, the value of the class misclassification function is a good indicator if a correct classification decision has been made and how far it is from the decision boundary (a good measure of robustness). At the beginning of MFoM learning, there is a large overlap between the distributions of the *C*+ and *C*- samples. This implies a higher error rate. After 500 iterations of MFoM learning the curve of the positive class moves left while that of the negative class moves right. This results in a smaller overlap and a reduced error rate. Figure 3 also shows that the curves of the distribution become 'flat' after MFoM training, a clear indication that the MFoM-trained classifier is more robust and less sensitive to data variation.

Recall that SVM tries to find a decision boundary that gives a maximal distance between the two classes. MFoM learning tries to do the same and directly optimizes the desired performance metrics. The measures observed in training could also be used to predict the performance for the unobserved testing data.

7. SUMMARY AND CONCLUSION

In this paper we propose a maximal figure-of-merit (MFoM) learning framework, in which an overall objective function is designed to directly relate the parameters of the classifier to the performance metrics of interest. A smooth approximation of some discrete quantities representing error counts is required to embed the classifier decision rules into the objective function. This decision-feedback learning framework is attractive because it offers a novel way to directly optimize the performance of any classifier with any evaluation measures of interest. These evaluation measures obtained in training could also be used to predict the same metrics computed on a similar testing set, making it easy for a designer to estimate the performance of the classifier without the need of running an extensive set of experiments or collecting a large set of evaluation data, which could be very expensive.

Using the ModApte version of the Reuters-21578 TC task, we first studied two different MFoM learning methods, namely MCE-based MFoM, using error rates, and F_1-based MFoM, using F_1, as the training objectives. The latter gives better micro-averaging F_1. We then compared the MFoM learning approach with a baseline binary classifier and a linear SVM classifier. The results showed that the F_1-based MFoM learning approach with the full rank (1,613) LSI features outperformed SVM with 9,600-dimension features. It also enhanced the robustness and improved the performance over the baseline classifier. Using only 400 LSI features, the MFoM approach achieved the micro-averaging F_1 value of 0.8702, comparable to that obtained with the popular SVM classifier.

Finally we have also demonstrated how MFoM learning could be extended to multi-category classification. Similar to the MCE-formulation used in many applications, such as speech recognition, a one-dimensional misclassification function could be defined to measure the degree of separation between the correct and all the competing classes collectively. An overall objective function could then be defined accordingly to embed the classification decision rules into the training objective for optimization. This offers a new tool for designing high performance, multi-category classifiers for many new applications.

We anticipate more future work on MFoM learning, including a comparative study on the evaluation of different performance metrics using different training objectives on individual classes and the overall design. We will examine MFoM-based multi-category classifiers beyond the currently prevailing binary classifiers. We will also extend the MFoM methodology to other interesting classification and verification problems in natural language processing, text categorization, information retrieval and data mining.

REFERENCE

[1] Bellegarda, J.R., Exploiting latent semantic information in statistical language modeling. In Proceedings of the IEEE, Vol.88, No.8, Aug. 2000.

[2] Berry, M. et al., SVDPACKC (Version 1.0) User's Guide, *Univ*. of Tennessee Tech. Report CS-93-194, April 1993.

[3] Breiman L., J. Friedman H., Olshen R.A., and Stone C.J., Classification and Regression Trees, Wadsworth Int. 1984.

[4] Brodley, C.E. and Utgoff, P.E., Multivariate decision trees, In COINS Technical Report 92-82, Dec., 1992.

[5] Buckley, C., Salton, G. and Allan, J., The effect of adding relevance information in a relevance feedback environment, In ACM SIGIR'94, 1994.

[6] Cortes, C., and Vapnik, V., Support vector networks, In Machine Learning, 1995.

[7] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K. and Harshman, R., Indexing by latent semantic analysis, In Journal of the American Society for Information Science, 41(6): 391-407, 1990.

[8] Denoyer L., Zaragoza H., and Gallinari P., HMM-based passage models for document classification and ranking, In ECIR'01, 2001.

[9] Duda, R. O., Hart, P. E. and Stork, D. G., *Pattern classification*, Second Edition, A Wiley-Interscience Publication, 2001.

[10] Guo, H. and Gelfand S.B., Classification trees with neural network feature extraction, In IEEE Trans. on Neural Networks, Vol. 3, No. 6, Nov., 1992.

[11] Hofmann, T., Probabilistic latent semantic indexing, In ACM SIGIR'99, 1999.

[12] Joachims, T., A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization. In ICML'97, 1997.

[13] Joachims, T., Learning to classify text using Support Vector Machines. Kluwer Academic Publishers, 2002..

[14] Joachims, T., Text categorization with Support Vector Machines: learning with many relevant features, In ECML'98, Springer, 1998.

[15] Juang, B.-H., Chou, W., and Lee, C.-H., Minimum classification error rate methods for speech recognition, In IEEE Trans. Speech and Audio Processing, Vol.5, No. 2, pp.257-265, Mar, 1997.

[16] Katagiri, S., Juang, B.-H. and Lee, C.-H., Pattern recognition using a family of design algorithm based upon the generalized probabilistic descent method, In Proceedings of the IEEE, Vol. 86, No. 11, 1998.

[17] Kuo, H.-K.J. and Lee, C.-H., Discriminative training of natural language call routers, to appear in IEEE Trans. Speech and Audio Processin*g*, 2003.

[18] Lee, C.H. and Huo Q., On adaptive decision rules and decision parameter adaptation for automatic speech recognition, In Proceedings of the IEEE, Vol.88, No.8, August 2000.

[19] Lee, C.H., Soong, F.K. and Paliwal, K.K., Automatic Speech and Speaker Recognition: Advanced Topics, Kluwer Academic Publishers, Boston, 1996.

[20] Lewis, D. and Gale, W., A comparison of two learning algorithms for text categorization. In The Third Annual Symposium on Document Analysis and Information Retrieval, pp.81-93, 1994.

[21] Li, Y. H., and Jain, A.K., Classification of text documents, In The Computer Journal, 41(8):537-546, 1998.

[22] Liu, J.M. and Chua, T.S., Building semantic perceptron net for topic spotting, In ACL'01, July 2001.

[23] Makato, I. and Takenobu T., Cluster-based text categorization: a comparison of category search strategies, In ACM SIGIR'95, 1995.

[24] Masand, B., Lino G., and Waltz, D., Classifying news stories using memory based reasoning, In ACM SIGIR'92, 1992.

[25] McCallum, A., Nigam, K., A comparison of event models for naïve Bayes text classification, In AAAI-98 Workshop on Learning for Text Categorization. Tech. Rep. WS-98-05, AAAI Press, 1998.

[26] Miller, D. R. H., Leek T., and Schwartz, R. M., A Hidden Markov model information retrieval system, In ACM SIGIR'99, 1999.

[27] Ng, H. T., Goh, W. B., and Low, K. L., Feature selection, perceptron learning, and a usability case study for text categorization, In ACM SIGIR'97, 1997.

[28] Quinlan, J., C4.5: Programming for machine learning, Morgan Kaumann, 1993.

[29] Ruiz, M.E. and Srinivasan, P., Hierarchical neural networks for text categorization, In ACM SIGIR'99.

[30] Schapire, R., Singer, Y., BoosTexter: A boosting-based system for text categorization, In Machine Learning, 2000.

[31] Schutze, H., Hull, D.A., and Pedersen, J. O., A comparison of classifier and document representations for the routing problem, In ACM SIGIR'95, 1995.

[32] Sebastiani, F., Machine learning in automated text categorization, In ACM Computing Surveys, Vol.34, No.1, pp.1-47, March 2002.

[33] Utgoff, P.E. and Brodley, C.E, Linear machine decision trees, In COINS Technical Report, 91-10, 1991.

[34] Vapnik, V., The Nature of Statistical Learning Theory, Springer-Verlag, 1995.

[35] Wu, H., Phang T., Liu, B., and Li, X., A refinement approach to handling model misfit in text categorization, In KDD'02, July 2002.

[36] Yang Y. and Liu X., A re-examination of text categorization methods, In ACM SIGIR'99, 1999.

[37] Yang, Y., Expert network: Effective and efficient learning from human decisions in text categorization and retrieval, In ACM SIGIR'94, 1994.